IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant : Mark E. Kriegsman and   Art Unit : 2141
            Benjamin W. Wyckoff   Examiner : Djenane M. Bayard
Serial No. : 09/668,110
Filed : September 22, 2000
Title : SERVING DYNAMIC WEB-PAGES

Mail Stop Appeal Brief - Patents

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

DECLARATION OF BENJAMIN W. WYCKOFF UNDER 37 C.F.R. 1.131

I, Benjamin W. Wyckoff, hereby declare:

1.	That I am a co-inventor of the claims in the above-captioned patent application.

2.	That in an Office Action dated July 18, 2006, certain claims were rejected as being unpatentable over *Ims*, et al., US2004/0230747, which has a priority date of July 6, 2000.

3.	That on or before July 6, 2000, in the United States, a WTO member country, we conceived a method for enabling the generation of an updated web-page for storage in one of a plurality of cache servers, the method comprising: implementing programmable rules executing on each of the plurality of cache servers, each programmable rule defining a triggering event associated with its corresponding cache server, the occurrence of the triggering event being indicative of the existence of an obsolete portion of the web-page stored in the corresponding cache server; detecting an occurrence of a triggering event at a particular cache server selected from the plurality of cache servers; in response

to the occurrence of the triggering event, causing the particular cache server to request an update of the obsolete portion; and receiving an updated portion of the web-page for storage at the particular cache server.

4. That on or before July 6, 2000, in the United States, a WTO member country, we conceived of a computer-readable medium having encoded thereon software for updating web-pages stored in caches, each cache being associated with a corresponding cache server from a plurality of cache serves, the software comprising instructions for implementing programmable rules executing on each of the plurality of cache servers, each programmable rule defining a triggering event associated with its corresponding cache server, the occurrence of the triggering event being indicative of the existence of an obsolete portion of the web-page stored in the corresponding cache server; detecting an occurrence of a triggering event at a particular cache server selected from the plurality of cache servers; in response to the occurrence of the triggering event, causing the particular cache server to request an update of the obsolete portion; and receiving an updated portion of the web-page for storage at the particular cache server.

5. That on or before July 6, 2000, in the United States, a WTO member country, we conceived of a web-serving system comprising a plurality of cache servers each having a corresponding cache memory; and a cache manager in communication with the corresponding cache memory for controlling content of the corresponding cache memory, the cache manager being configured to execute a programmable script, the script being configured for detecting the occurrence of a triggering event, and in response to detection of the triggering event, causing the cache manager to request an update of the content of the cache memory.

5. That as evidence of my conception, I have enclosed herewith:

    Exhibit A, which is a copy of the first six pages of an email string provided to a private focus group of users who agreed to assist us in testing our invention, which we named "DBFX;"

Exhibit B, which is a copy of the first four pages of a document I prepared to describe architecture notes and design and implementation decisions for DBFX;

Exhibit C, which is a copy of the first four pages of a document I prepared to describe certain environmental variables, script tags, and attribute names used during processing of DBFX scripts; and

Exhibit D, which is an exemplary DBFX script.

6.   The following table sets forth support for the features of the invention claimed in independent claim 1 and the corresponding features in independent claim 18.

CLAIM LIMITATION	SUPPORT FOR CONCEPTION
A method for enabling the generation of an updated web-page for storage in one of a plurality of cache servers, said method comprising:	
implementing programmable rules executing on each of the plurality of cache servers, each programmable rule defining a triggering event associated with its corresponding cache server, the occurrence of the triggering event being indicative of the existence of an obsolete portion of said web-page stored in said corresponding cache server;	Exhibit A: page 6 "There's even a simple IF-ELSE-ENDIF syntax, so you can make up your own rules." Exhibit A: page 2, "DBFX is extremely flexible in letting you create intelligent cache-management rules that keep all your queries and databases in sync." Exhibit A, page 2: "DBFX lets you transparently forward database queries from your main server to a second server." Exhibit B: page 3, bottom of page "DBFX is controlled by real or synthesized "DBFX files" which contain instructions for modifying the request environment and specifying the action to take. Exhibit C: page 2, see syntax for "IF" at bottom of page.

detecting an occurrence of a triggering event at a particular cache server selected from the plurality of cache servers;	This would be whatever condition is specified in any of the "IF" statements above. For example, see Exhibit A: page 5, "You can easily set up any combination of: - specify a fixed amount of time to cache". See also Exhibit D sample "if" statements.
in response to the occurrence of said triggering event, causing said particular cache server to request an update of said obsolete portion; and	Exhibit D "decache" command executed. See also Exhibit A, page 5 referring to pre-cache and de-cache instructions, as well as ability to include or exclude any subset of queries from DBFX caching or processing. Exhibit B, page 4 "The [DBFX] file may also specify that DBFX should retrieve the data, flush the data, issue a redirect, etc."
receiving an updated portion of said web-page for storage at said particular cache server.	See Exhibit A page 5. This is the natural consequence of the retrieve option specified in Exhibit B, page 4.

7.   The following table sets forth support for the features of the invention claimed in independent claim 13.

CLAIM LIMITATION	SUPPORT FOR CONCEPTION
A web-serving system comprising:	
a plurality of cache servers each having a corresponding cache memory; and	Prior art, needs no evidence of conception.
a cache manager in communication with said corresponding cache memory for controlling content of said corresponding cache memory, said cache manager being configured to execute a programmable script, said script being configured for detecting the occurrence of a triggering event, and in response to detection of said triggering event, causing said cache manager to request an update of said content of said cache memory.	Cache manager is whatever executes the script referred to in claim 1.

8. That on or before July 6, 2000, in the United States, a WTO member country, we reduced our invention to practice by writing software for carrying out the method steps recited in claim 1, encoding the software into a computer readable medium as recited in claim 19, and testing it on a system as recited in claim 13.

9. The dates in Exhibits A, B, C, and D have been redacted, however all the redacted dates were on or before July 6, 2000.

10. I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true. I further declare that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, under Section 1001 of Title 18 of the United States Code, and that such willful false statements may jeopardize the validity of the application or any patent issuing thereon.

_January 17, 2007_____   _/Benjamin W. Wyckoff/_
Date   Benjamin W. Wyckoff

WELCOME!

Welcome to the private Focus Group for ClearWay Technologies'
new accelerator for Web database publishing, codenamed "DBFX"

DBFX speeds Web database queries, and gives you powerful,
and flexible new options for how you publish your databases
on the Web.

YOU ARE NOW PART OF THE FOCUS GROUP

You have been added to the DBFX-Beta mailing list because
you asked to be added, or because you expressed an interest
in a product that accelerates Web database publishing.

This small, closed mailing list will be used to tell you
where to download the new software, and for discussions
of how to use it for your database-publishing applications.

NON-DISCLOSURE

By electing to participate in this focus group, you will be in
possession of proprietary, confidential, and trade secret information
belonging to ClearWay Technologies, Inc. You must not disclose
the content, or the nature, or character of the information discussed
here to any third party without prior written consent from
ClearWay Technologies, Inc. The sole exception is information
that you are in prior possession of, or that becomes public through
other means. You will be held liable for all damages arising

directly or indirectly from your actions if you disclose protected information

If you cannot abide by this plain-language non-disclosure agreement, contact me at once at <kriegsman@clearway.com> and you will be removed from the focus group and this mailing list.

WHAT THE DBFX SOFTWARE DOES

DBFX provides four new functions for your Web database:

1. Caching. The results of the most frequently/recently requested queries are cached, greatly speeding publishing applications using a database. DBFX is extremely flexible in letting you create intelligent cache-management rules that keep all your queries and databases in sync.

2. Virtual Queries. DBFX lets you create virtual URL hierarchies that actually translate into database queries. This allows you to hide the structure and nature of your database system, and can allow Web crawlers to index your databased content.

3. Forwarding. DBFX lets you transparently forward database queries from your main server to a second server, via TCP/IP. This allows you to move your database to a second computer without exposing the 'inner' structure to the outside world.

4. Data Insertion. DBFX lets you embed data from your databases on any NetCloak or WebSTAR SSI page with a few simple tags. Thus, you could use SSI for a rotating-ad-banner system, and still serve Web content from your database through the extremely fast DBFX cache.

DBFX works with: Lasso, Tango, WebCatalog, WebFM, FileMaker, and other popular Web databases. It's a WebSTAR plug-in, and will also be available for WebTen and other W*API-compatible servers. An Apache version is also in the works.

That's it in a nutshell. If you are doing "Web Database Publishing", then DBFX will probably make your job easier, and you Web site run much faster.

If you are using a Web database only for a highly-interactive
application (such as a shopping cart system), DBFX will not
accelerate it, but you may want to look at the other features as well.

HOW AND WHEN DO I GET THE SOFTWARE?

The software will be available for download later this week. You
will be informed on this mailing list.

We expect every member of the focus group to be an active participant
in this discussion and the refinement of DBFX. If you cannot
spare the time (a few hours a week, for about four weeks), or if
you will not have any opportunity to try DBFX, or if you simply
don't want to do this now, just drop me an e-mail.

DON'T WANT TO BE PART OF THE FOCUS GROUP?

If you wish to be removed from the focus group, e-mail me directly
at <kriegsman@clearway.com> You cannot remove yourself.

OK- Welcome to the list, and thank you for helping us out.

I hope that all this sounds interesting to you, and if it does, I'm
sure you have a million questions... anyone want to kick off the
discussion based on the four features identified above (Caching,
Forwarding, Virtual Queries, Data Insertion) ?

I'm waiting for "how does it know how long to keep a query cached?" !

-Mark Kriegsman
 kriegsman@clearway.com
 1-617-262-4006

From ???@???   16:22:44
Received: from 205.136.66.214 by clearway.com

with SMTP (Eudora Internet Mail Server 1.2); ▓▓▓▓▓▓▓▓ 16:19:17 -0500
Date: ▓▓▓▓▓▓▓▓ 22:11:31 +0100
Message-Id: <v04103d05b2c953ab3f7c@[195.198.39.100]>
From: "Joakim Jardenberg [listmail]" <inbox@infinit.se>
Subject: Re: Web database accelerator (DBFX) Beta list
To: dbfx-beta@clearway.com
Mime-Version: 1.0
Content-Type: text/plain ; format="flowed"
Precedence: Bulk
X-Listserver: ListSTAR v1.1 by StarNine Technologies, a Quarterdeck Company
Reply-To: dbfx-beta@cleaway.com
Errors-To: dbfx-beta@cleaway.com

▓▓▓▓▓ klockan 16.03 -0500, skrev Mark Kriegsman:
> I hope that all this sounds interesting to you, and if it does, I'm
> sure you have a million questions... anyone want to kick off the
> discussion based on the four features identified above (Caching,
> Forwarding, Virtual Queries, Data Insertion) ?

how does it know how long to keep a query cached?

;-)

Very excited about the next CWKA (Clearway killer app)

Also, note, the return address i missing the "r" in clearway
(dbfx-beta@cleaway.com)

/Jocke

--
Joakim Jardenberg: http://www.infinit.se/.sig

innehŒllet i detta brev fŒr inte utan sŠrskilt
medgivande publiceras eller pŒ annat sŠtt spridas

From ???@???, ▓▓▓▓▓▓ 6:42:00 ▓▓▓
Received: from 205.136.66.214 by clearway.com
 with SMTP (Eudora Internet Mail Server 1.2); ▓▓▓▓▓▓▓▓ 6:43:43 -0500
Date: Mon, ▓▓▓▓▓▓ 16:36:27 -0500
Message-Id: <v02140b06b2c959c6e323@[205.136.66.194]>
From: kriegsman@clearway.com (Mark Kriegsman)
Subject: Re: Web database accelerator (DBFX) Beta list
To: dbfx-beta@clearway.com
Mime-Version: 1.0
Content-Type: text/plain; charset="us-ascii"

4

At 10:11 PM on █████ Joakim Jardenberg [listmail] wrote:

: █████klockan 16.03 -0500, skrev Mark Kriegsman:
> > I hope that all this sounds interesting to you, and if it does, I'm
> > sure you have a million questions... anyone want to kick off the
> > discussion based on the four features identified above (Caching,
> > Forwarding, Virtual Queries, Data Insertion) ?
>
> how does it know how long to keep a query cached?
>
> ;-)

Good question! You can configure DBFX to cache queries as long (or as
short, or as intelligently) as you wish.

You can easily set up any combination of:

- specify a fixed amount of time to cache (can vary per query, and globally)
  such as "one hour by default, but only 5 minutes for one special page"

- specify that when a query says "action=insert" (or anything else you wish),
  that DBFX will then flush any cached records from that same database.
  This lets you 'trap' any INSERT, UPDATE, and DELETE events that happen
  through Web-based interfaces.

- de-cache and/or pre-cache certain queries when the _Web_server's_ caches
  are flushed

- respect or ignore the "Expires" date in the original query result

- respect or ignore any No-cache directives in the browser request

- de-cache and/or pre-cache any queries you wish when a specific URL is
  requested (and you can defined any number of URLs for controlling the
  cache any way you wish...)

- include or exclude any set or subset of your queries from DBFX
  caching or processing; DBFX only caches queries you designate

5

There's even a simple IF-ELSE-ENDIF syntax, so you can make up your own rules.

Hopefully, this will all be clearer with the software in front of you. But
the general concept is that you can specifify with great ease and
flexibility how long you want each query cached.

And thanks for asking, Joakim.

-Mark

〜〜〜〜〜〜〜〜〜〜〜〜〜〜〜〜〜〜〜〜〜〜〜〜〜〜

From ???@??? ▬▬▬ 17:12:39 ▬▬▬
Received: from 205.136.66.214 by clearway.com
 with SMTP (Eudora Internet Mail Server 1.2); ▬▬▬▬ 17:08:03 -0500
Date: Mon, ▬▬▬▬ 22:50:47 +0100
Message-Id: <v04103d01b2c95c915842@[195.198.39.100]>
From: "Joakim Jardenberg [listmail]" <inbox@infinit.se>
Subject: Re: Web database accelerator (DBFX) Beta list
To: dbfx-beta@clearway.com
Mime-Version: 1.0
Content-Type: text/plain ; format="flowed"
Precedence: Bulk
X-Listserver: ListSTAR v1.1 by StarNine Technologies, a Quarterdeck Company
Reply-To: dbfx-beta@clearway.com
Errors-To: dbfx-beta@clearway.com

▬▬▬ klockan 16.36 -0500, skrev Mark Kriegsman:
> There's even a simple IF-ELSE-ENDIF syntax, so you can make up
> your own rules.

WOW!

That is definetly more advanced then what I was looking for on
Lasso-Talk almost a year ago, I quote myself:

- - -

As one who is a sucker for "semi-dynamic" publishing solutions a
LARGE cache in Lasso would be great.

A typical url in one of our systems point to a formatfile with a
inline that uses the form_param to retrive the article (ie:
http://www.mydomain.com/display.lasso?show=136

B

This document contains some architecture notes and design
and implementation decisions for DBFX.

Server/Client Model Theory

(This is classic middleware)

DBFX is based internally on a Server/Client Model. The primary
task of DBFX is to SERVE pages to clients making requests to the
web server. Internally, DBFX turns around and acts as a CLIENT,
getting data from a variety of other sources.

As a server, DBFX may get requests in a number of ways, including,
but not limited to, as a W*API Plug-In in the CGI role, as a PIXO
source or processor, and as a W* SSI Tag handler.

As a client, DBFX may get data in a number of ways, including,
but not limited to, PIXO to other plug-ins running in the
same server, TCP/IP to the same server or other servers
(running anywhere), W* SSI specific services, Apple Events to
CGIs, FastCGI.

The internal architecture turns the various requests contexts
into a normalized request parameter block, with most of the
data stored in an Environment data structure (PassiveVarEnv).
Similarly, returning data back to the clients is hidden through
use of an abstract Emitter object. This way, regardless of the
request method (e.g., PIXO, W*API CGI), we can process the request
in the same way after normalizing it. The REQUEST code is new
to DBFX, but based on existing code, technology, and concepts.

The client code is handled in a similar way, with much of the
code borrowed from the FireSite client technology. An HTTPRequest
class (and its derived classes) provides a uniform interface to
performing HTTP requests and handling the responses. Current
clients support TCP/IP and PIXO. Extending the functionality to
include AppleEvents or other methods is relatively straightforward,
and does not alter the architecture of the system at all.

1

Server/Client Model Implementation

This section describes the architecture in terms of the specific
classes and flow of control.

Request classes:
        class DBFXRequestPB (virtual base class)
                class DBFXCGIRequestPB : public DBFXRequestPB
                class DBFXPIXORequestPB : public DBFXRequestPB
                class DBFXSSIRequestPB : public DBFXRequestPB

Emitter classes:
        (These may change, depending on how the cache is implemented)
        class SimpleEmitter
                class WSAPIEmitter : public SimpleEmitter
                class BufferedEmitter : public SimpleEmitter

Client classes:
        (There are a few levels here, to take advantage of existing code)
        class DBFXClient
                class DBFXSimpleHTTPClient : public DBFXClient (prototype/test class)
                class DBFXHTTPRequestClient : public DBFXClient

Client classes, level 2:
        (class DBFXHTTPRequestClient manages different HTTPRequest derived classes)
        class HTTPRequest
                class HTTPRequestTCP : public HTTPRequest
                class HTTPRequestPIXO : public HTTPRequest

The basic flow is as follows:

Request_Entry_Point
        custom entry point processing
        create appropriate derived class of DBFXRequestPB
        call DBFXRequestPB.ProcessRequest()
                InitializeEnv(serverField);
                PopulateEnv();
                RunDBFXInterpreter();
                Execute(0);
                        // Note - cache hits will (probably) be processed here
                        based on Env, prepare a new request, allocate new client class,
                        execute the request, and the result is written to the emitter
                        on the DBFXRequestPB.
                Emitter().flush();

2

```
                // Note - cache insertion will (probably) happen here.
        perform custom request cleanup
                (may include actually handing data back to client, if not streaming)
        custom entry point exit-processing


Some specific entry points that are currently implemented are
as follows:


PIXO Server:
PIXO_RunResult PIXOEntry(void * pixoPB);
        cast pixoPB to appropriate type
        create thread context
        call PIXOAction(pixoPB)
                if(status request)
                        return OKToRun
                else {
                        get server header field for responses
                        DBFXPIXORequestPB dbfx(pixoPB);
                        dbfx.ProcessRequest(serverField);
                        // the response is buffered by the Emitter, because PIXO works in "page
mode"

                        pixoResult = dbfx.StoreResultOnPIXOPB();
                }
                return pixoResult
        handle exceptions
        return pixo_result


W*API CGI Role Server:
WSAPI_Dispatch
        if(WSAPI_Run) {
                ProcessRunCall (commandPtr);
                        get server header field for responses
                        if admin request, handle directly
                        DBFXCGIRequestPB dbfx(pb);
                        dbfx.ProcessRequest(serverField);
                        // the response is streamed back to the client by the Emitter as it is
generated
                        return WSAPI_I_NoErr;

        }


------------------------------------------------------------
DBFX "Command" Files


DBFX is controlled by real or synthesized "DBFX files" which contain
instructions for modifying the request environment and specifying the
action to take.
```

Specifically, a DBFX file might set the host name, scriptname, path and search args describing the location of the desired content. The file may also specify that DBFX should retrive the data, flush the data, issue a redirect, etc.

As a DBFX server receives a request, it populates an environment with the request parameters. The original request may identify a physical DBFX file on disk. If so, the file contents are parsed, with the tag-handler side effects possibly modifying the request environment. Also, the original request (e.g., for PIXO or SSI) may include DBFX commands in the request, so there is no physical file to load. In this case, the commands are similarly processed for side effect. Lastly, there may be global rules applied to the request, which may result in some other DBFX file or command set to be processed. For instance, a "virtual URL" may not refer to a real file, but may contain information that points to an alternate DBFX command file.

In all cases, the request environment may be modified by commands in a real or virtual DBFX command file.

After the DBFX command file has been processed, the environment is handed off to the DBFX Client manager.

(Note - the DBFX command "file" is processed during the RunDBFXInterpreter phase of server request handling)

Cache

There are no cache design notes at this time, as the design is still in flux.

C

DBFX Env Variables.txt
Ben Wyckoff
█████████████
Updated████████

This document contains names (and some semantics) of the environment
variables , DBFX Script Tag and attribute names, and SSI Tag Handler
tag and attribute names used during DBFX processing.

It is intended that this document be reviewed with an eye towards
renaming many of the symbols to improve overall clarity.

There are three broad categories of symbol names that the end user
sees: DBFX Script Tag and Attribute names (DBFX Tag Names),
DBFX Script Variable Names (DBFX Var Names), and DBFX SSI Tag Handler
Tag and Attribute Names (SSI Tag Names). The DBFX Var Names are further
broken down into categories. (Yet another class of symbols is constants
- I'll lump them in with DBFX Var Names...)

The naming conventions should be similar, but they are discussed
separately below.

SSI Tag Names

DBFX registers "tag handlers" with WebSTAR SSI, and SSI calls the
registered function when it encounters the specified tags. All of
the attribute/value pairs in the tag are passed through to DBFX
but some of them have special semantics and are copied/coerced into
other internal environment variables.

Online documentation is currently at <http://www.clearway.com/dbfx/docs/SSI.html>

Tag Names:

EXEC_DBFX
EXEC_DBFX_SSI   - like EXEC_DBFX, except SSI processes the results

The above two tags take all of their parameters as attribute/value pairs - there
is no closing tag.

1

EXEC_DBFX_SCRIPT and /EXEC_DBFX_SCRIPT
EXEC_DBFX_SCRIPT_SSI and /EXEC_DBFX_SCRIPT_SSI

The _SCRIPT tag names do accept attribute/value pairs, but also interpret the
text between the begin and end tags as the DBFX script to be executed. The
_SSI variant causes SSI to process the results for additional SSI tags.

The EXEC_DBFX* tag handler sets all of the attribute/value pairs in the environment
as "FORM.attrname" variables. Some of the attributes are special cases as defined
below:

PATH="dbfx_file_name"   - the "SCRIPT_NAME" of the file-based DBFX script to execute.
        (does not apply to the _SCRIPT variants)

STARTBEFORE="delimiter_text"
STARTAFTER="delimiter_text"
ENDBEFORE="delimiter_text"
ENDAFTER="delimiter_text"

The start/end attributes are used internally to control how much of the entity is
returned to SSI from the tag handler. Note that the start/end variables may be set
directly in the _SCRIPT variants using <SET STARTBEFORE="..."> commands in the
script body as well as passed as attribute/value pairs in the tag body.

DBFX Tag Names

The DBFX Script Language is documented online at

<http://www.clearway.com/dbfx/docs/ScriptReference.html>

<DISPLAY_MESSAGE "text">

<SET name = expression>

<IF expression> ... <ELSE> ... <ENDIF>
(also </IF> as synonym for <endif>)

<DECACHE "match_string">

2

```
<DECACHE_REGEXP "regular_expression_string">
```

```
<PRECACHE HREF="url" METHOD="method" POSTARGS="post_arg_string" ...>
```
The ... means you can also specify arbitrary attribute/value pairs in the
PRECACHE tag. DBFX assigns the value into a variable with the same name as
the attribute (NOT as FORM.attrname). This is particularly useful for specifying
cache-control values for the individual precache request, such as DBFX.CACHE.TTL.

```
<INSERT_CONTENT HREF="url" METHOD="method" POSTARGS="post_arg_string" ...>
```
The ... means the same thing as in PRECACHE. In addition the START/END tags
used in EXEC_DBFX are valid here as well.

```
<RETRIEVE_CONTENT HREF="url" METHOD="method" POSTARGS="post_arg_string"
...>
```
This is like INSERT_CONTENT (including the START/END tags), plus the
"INTOVAR" attribute name is special - it tells DBFX what env variable to store
the result in.

```
<INSERT_STRING "value">
```

```
<REDIRECT HREF="url">
```
Internally (i.e., not documented), DBFX also accepts "URI", "URL", "LOCATION" as
synonyms for the "HREF" attribute name.

```
<ON_ERROR ...>
```
// attribute names are not yet documented online...
```
<ON_ERROR ACTION= "REDIRECT" | "SERVE_FROM_CACHE"
        URI=xxx>
```
Synonyms for URI are "URL" and "LOCATION"

```
<SYSBEEP>
```

```
<MATCH_REGEXP ...>
```
// attribute names are not yet documented online...
// in fact, the regexp behavior and is not yet documented online either
C4-based attributes: "SRC","REGEXP", "INTO",
DBFX custom attr "IGNORECASE" - causes DBFX to (incorrectly) ignore case

when doing the regexp match - defaults to "FALSE"
INTO defaults to "MATCH.", and defines the prefix for the regexp match vars

<INCLUDE "path">
(DBFXINCLUDE is a synonym for INCLUDE)

<DEFINE_SITE UINAME="name" PATH="siteroot" SCRIPTSPATH="scriptsfolder"
        HOSTNAMES="hostname_list">

<REPEAT> and </REPEAT>
// completely undocumented in DBFX (but identical to C4 usage)
// needed for Admin, might as well document as part of the DBFX Script Language.

DBFX Var Names

This major section encompasses DBFX script environment variable
names and values. There are 3 main types of DBFX scripts:
Automatic scripts, Filter Scripts, and Request Scripts.

Automatic scripts include startup, shutdown, flush, every-minute etc
and precache. The STARTUP and SHUTDOWN scripts run in the global
environment by design. All other scripts run in a temporary env that
has the global env as a parent.

Filter Scripts run at W*API filter time, and do not generate content.

Request Scripts run at "CGI" time, and are responsible for directly
or indirectly generating content to return to the client.

Global Variables

Cache Control

(documented online at http://www.clearway.com/dbfx/docs/CacheControl.html)

There are global variables that determine default cache behavior, and
there are variables that can modify cache behavior on a per-request or
per-entity basis.

```
<!--
        BWW ⬤⬤⬤⬤
        this script is run when DBFX can't find the requested
        file, and the file it tried to find ended in ".lasso.dbfx"

        This assumes your lasso URLs contain one of
                -Search=...
                -Update=...
                -Add=...
                -Delete=...

        If Update, Add, or Delete, decache all cached pages.
        The process the request.
-->

<if "[form.-update]" ne "">
<decache "/">
<endif>

<if "[form.-add]" ne "">
<decache "/">
<endif>

<if "[form.-delete]" ne "">
<decache "/">
<endif>


<!--
        Remove the last suffix (which is ".dbfx" now) from the script_name variable.
        The script_name here is the URL DBFX will request and cache
-->
<set script_name="[path.withoutsuffix.-0]">
<display_message "virtual dbfx script restoring script_name to [path.withoutsuffix.-0]">
```